robotron

Anleitung für den
Bediener

Symbolischer Debugger SID 86

Arbeitsplatzcomputer A 7100 Betriebssystem SCP 1700

Anleitung für den Bediener

Symbolischer Debugger SID86

AC A7100

VEB Robotron-Projekt Dresden

C 1015-0004-1 M 3030

Die vorliegende Systemunterlagendokumentation, Anleitung für den Bediener Symbolischer Debugger SID86 , entspricht dem Stand von 6/86.

Nachdruck, jegliche Vervielfältigung oder Auszüge daraus sind unzulässig.

Die Ausarbeitung erfolgte durch ein Kollektiv des VEB Robotron-Elektronik Dresden .

Im Interesse einer ständigen Weiterentwicklung werden alle Leser gebeten, Hinweise zur Verbesserung dem Herausgeber mitzuteilen.

Herausgeber:

VEB Robotron-Projekt Dresden 8010 Dresden, Leningrader Str. 9

(C) VEB Robotron 1986

Kurzreferat

SID86 ist ein leistungsfähiges Dienstprogramm des Betriebssystems SCP 1700. SID86 ermöglicht den Test und die Korrektur von Maschinencodeprogrammen. SID86 bietet gegenüber dem Debugger DDT86 besondere Leistungen. Dazu gehören symbolische Adreßbezüge, permanente Protokollierpunkte und Protokollierung (Trace) ohne Unterprogrammrufe.

Inhaltsverzeichnis Seite

1.	Arbeit mit SID86	5
1.1.	Aufruf	5
1.2.	Kommandoformen	6
1.3.	Angabe einer 20-bit-Adresse	7
1.4.	Interrupt-Behandlung	7
2.	Ausdrücke in SID86	8
2.1.	Hexadezimale Literale	8
2.2.	Dezimale Literale	8
2.3.	Alphanumerische Literale	9
2.4.	Registerwerte	9
2.5.	Stack-Bezüge	10
2.6.	Symbolische Bezüge	10
2.7.	Ausgewählte Symbole	11
2.8.	Operatoren in Ausdrücken	11
2.9.	Beispiele für symbolische Ausdrücke	12
3.	SID86-Kommandos	14
3.1.	A-Kommando (Assemble)	14
3.2.	B-Kommando (Block Compare)	15
3.3.	D-Kommando (Display)	15
3.4.	E-Kommando (Load Program, Symbols for Execution)	16
3.5.	F-Kommando (Fill)	18
3.6.	G-Kommando (Go)	18
3.7.	H-Kommando (Hexadecimal Math)	19
3.8.	I-Kommando (Input Command Tail)	20
3.9.	L-Kommando (List)	21
3.10.	M-Kommando (Move)	22
3.11.	P-Kommando (Pass-Point)	22
3.12.	R-Kommando (Read)	24
3.13.	S-Kommando (Set)	26
3.14.	T-Kommando (Trace)	27
3.15.	U-Kommando (Untrace)	29
3.16.	V-Kommando (Value)	29
3.17.	W-Kommando (Write)	30
3.18.	X-Kommando (Examine CPU-Status)	30
4.	Standardsegmentwerte	33
5.	Assemblersprache für A- und L-Kommandos	35
6.	Beispiel für einen Dialog mit SID86	36
Anlage	Fehlermitteilungen von SID86	48
Sachwortverzeichnis		50

Tabellenverzeichnis Seite

Tabelle 1:	SID86-Kommandos	6
Tabelle 2:	Kommandos und Standardsegmentwerte	34

0 1015-0004-1 M 3030

1. Arbeit mit SID86

SID86 (Symbolic Instruction Debugger) ist ein leistungsfähiger symbolischer Debugger. Er arbeitet unter dem Betriebssystem SCP 1700. SID86 hat gegenüber dem Standard-Debugger DDT86 des SCP 1700 erweiterte Arbeitsmöglichkeiten. SID86 ermöglicht dem Anwender, Programme unter SCP 1700 interaktiv zu testen und Fehler zu beseitigen. Zusätzlich kann symbolisch assembliert und reassembliert werden. Die entsprechenden Ausdrücke können hexadezimal, dezimal, Zeichenketten oder symbolische Werte sein. Außerdem können permanente Protokollierpunkte gesetzt werden und eine Protokollierung (trace) ohne Unterprogrammrufe ausgeführt werden. Der Leser sollte mit dem Betriebssystem SCP 1700 und der Assemblersprache ASM86 vertraut sein.

1.1. Aufruf

Der Aufruf von SID86 erfolgt durch eines der folgenden Kommandos:

```
(a) SID86
(b) SID86 filespec
(c) SID86 filespec {,symfile ...}
(d) SID86 * symfile {,symfile}
```

Das Kommando (a) lädt und startet SID86. Nachdem die Programmeldung und die Kommandoanforderung (#) erfolgte, ist SID86 zur Annahme von Kommandos bereit.
Die Form (b) ist analog zur Form (a), nur daß SID86 noch zusätzlich die durch filespec spezifizierte Datei lädt. Falls der Dateityp in filespec fehlt, wird CMD angenommen. SID86 kann keine Datei vom Typ H86 laden.
Das Kommando (c) hat die gleiche Wirkung wie (b), nur daß SID86 noch zusätzlich zum Programm die durch symfile spezifizierten Symboldateien in seine Symboltabelle lädt.
Die Form (d) ist ähnlich der Form (c), es wird aber kein Programm geladen, sondern nur die Symboldateien.
Die Formen (b), (c) und (d) sind analog den Kommandofolgen:

```
A>SID86
SID86 Vx.x
#Efilespec
```

```
A>SID86
SID86 Vx.x
#Efilespec {,symfile ...}
```

```
A>SID86
SID86 Vx.x
#E* symfile
```

Das Programm, das durch die Kommandoformen (b) oder (c) geladen wurde, ist danach zur Ausführung bereit. (Zur Beschreibung des E-Kommandos siehe Abschnitt 3.4.).

1.2. Kommandoformen

Die Kommandoanforderung von SID86 ist das Zeichen #. Als Kommando kann der Bediener eine Kommandozeile oder ein CTRL/C zum Beenden der Arbeit eingeben. Eine Kommandozeile kann bis zu 64 Zeichen lang sein und wird mit Wagenrücklauf (CR) abgeschlossen. Bei der Eingabe der Kommandozeile können auch die Korrekturfunktionen (CTRL/X, CTRL/H, CTRL/R usw.) zur Korrektur von Eingabefehlern genutzt werden. SID86 bearbeitet eine Kommandozeile erst nach dem Abschluß der Zeile durch Wagenrücklauf (CR).

Das erste Zeichen jeder Kommandozeile bestimmt das Kommando. Tabelle 1 enthält alle Kommandos von SID86.

Tabelle 1: SID86-Kommandos

Kommando	Wirkung
A	Eingabe von Assembleranweisungen
B	Vergleich zweier Speicherblöcke
D	Anzeige des Speicherblockes hexadezimal und KOI7
E	Laden eines Programms zur Ausführung
F	Füllen des Speicherblockes mit einer Konstanten
G	Testbeginn mit wahlweisen Unterbrechungspunkten
H	Hexadezimale Arithmetik
I	Aufstellen von Standard-FCB und Kommandoparameter
L	Auflisten des Speicherinhalts im Mnemonikformat
M	Umspeichern eines Speicherblockinhalts
P	Protokollierpunkte setzten, löschen oder anzeigen
R	Lesen einer Plattendatei in den Speicher
S	Prüfen und ändern des Speicherinhaltes
T	Programm-Ablaufverfolgung mit Protokoll (trace)
U	Programm-Ablaufverfolgung ohne Protokoll
V	Anzeige der Speicherbelegung der gelesenen Datei
W	Schreiben eines Speicherblockes auf die Platte
X	Prüfen und Ändern des ZVE-Status

Auf das Kommandozeichen können ein oder mehrere Argumente folgen. Das können symbolische Ausdrücke, Dateinamen oder andere Informationen in Abhängigkeit vom Kommando sein. Parameter werden voneinander durch Kommas oder Leerzeichen getrennt. Zwischen dem Kommandozeichen und dem ersten Parameter ist kein Trennzeichen zulässig. Ein Semikolon als erstes Zeichen kennzeichnet die Zeile als Kommentar. Verschiedene Kommandos (G, P, S, T und U) können mit einem Minuszeichen (-) beginnen. Die Wirkung des Minuszeichens ist von Kommando zu Kommando verschieden. Abschnitt 3. beschreibt die einzelnen Kommandos und die Wirkung des Minuszeichens.

1.3. Angabe einer 20-bit-Adresse

Die meisten Kommandos von SID86 erfordern eine oder mehrere Adressen als Operanden. Da der Prozessor bis zu 1 Megabyte Speicher adressieren kann, müssen die Adressen 20 bit lang sein. Eine 20-bit-Adresse wird wie folgt spezifiziert:

```
ssss:oooo
```

Dabei ist ssss eine optionale 16-bit-Segmentnummer und oooo ist ein 16-bit-Offset. SID86 verbindet diese Werte zu einer effektiven 20-bit-Adresse in der Form:

```
 ssss0
+ oooo
------
 eeeee
```

Die Angabe des Segmentwertes ist wahlweise. Falls der Segmentwert fehlt, benutzt SID86 einen für das Kommando geeigneten Standardwert. Die Standardsegmentwerte sind in Abschnitt 4. beschrieben.

1.4. Interrupt-Behandlung

SID86 arbeitet mit Interrupt-Behandlung und rettet den Interrupt-Status des Programms, das unter SID86 abgearbeitet wird. Wenn SID86 die Steuerung erhält, also wenn es gestartet wird oder die Steuerung von einem Testprogramm zurückerhält, dann ist das Interrupt-Flag so gesetzt, wie beim Aufruf von SID86, mit Ausnahme einiger kritischer Bereiche, in denen Interrupts verboten sind. Wenn das zu testende Programm die Steuerung der ZVE hat, dann bestimmt der ZVE-Status des Nutzers den Zustand des Interrupt-Flags. Dieser Zustand kann durch das X-Kommando angezeigt werden.

2. Ausdrücke in SID86

Ein besonderes Merkmal von SID86 ist die Möglichkeit, absolute Speicherplätze durch Ausdrücke zu adressieren. Ausdrücke können Symbole vom zu testenden Programm enthalten. Diese sind in der SYM-Datei definiert, die vom ASM86 erzeugt wird. Ausdrücke können auch aus Literalen in hexadezimaler, dezimaler oder Zeichenkettenform bestehen. Im Zusammenhang mit speziellen Operatoren kann so zu indexierten oder indirekt adressierten Daten- oder Programmbereichen zugegriffen werden. Dieser Abschnitt beschreibt die Form von Ausdrücken, wie sie als Kommandoparameter in den einzelnen Kommandos verwendet werden können.

2.1. Hexadezimale Literale

SID86 verarbeitet Werte in hexadezimaler Form. Zulässige hexadezimale Ziffern sind die dezimalen Ziffern 0 bis 9 und die Buchstaben A, B, C, D, E und F, die den dezimalen Werten von 10 bis 15 entsprechen.

Ein hexadezimaler Wert in SID86 besteht aus einer oder mehreren zusammenhängenden hexadezimalen Ziffern. Bei der Eingabe von Ziffern hat die erste Ziffer die höchste Wertigkeit. Wenn eine Zahl mit mehr als vier Ziffern eingegeben wird, dann sind die vier rechten Ziffern signifikant, die übrigen Ziffern links davor werden ignoriert. Die folgenden Beispiele zeigen die entsprechenden hexadezimalen und dezimalen Werte für die Eingabewerte.

Eingabe	Hexadezimal	Dezimal
1	0001	1
100	0100	256
FFFE	FFFE	65534
10000	0000	0
38001	8001	32769

2.2. Dezimale Literale

Die Eingabe eines dezimalen Literals beginnt mit dem Symbol #. Der folgende Wert muß aus einer oder mehreren dezimalen Ziffern (0 bis 9) bestehen. Die höchstwertige Ziffer steht am weitesten links. Dezimale Werte werden zu hexadezimalen Werten konvertiert und nach der Regel der hexadezimalen Literale mit Nullen aufgefüllt oder beschnitten.

Die Eingabewerte werden zu folgenen hexadezimalen Werten konvertiert:

Eingabe	Hexadezimalwert
#9	0009
#10	000A
#256	0100
#65535	FFFF
#65545	0009

2.3. Alphanumerische Literale

SID86 läßt ein oder zwei in Apostroph eingeschlossene KOI7-Zeichen als Literal in Ausdrücken zu. Die Zeichen werden unverändert übernommen, d. h. es erfolgt keine Konvertierung von Groß-/Kleinbuchstaben. Das erste Zeichen hat die höhere Wertigkeit. Zeichenketten aus einem Zeichen werden nach links mit Null aufgefüllt. Zeichenketten mit mehr als zwei Zeichen sind in Ausdrücken nicht zulässig. Eine Ausnahme ist im S-Kommando beschrieben.

Die begrenzenden Apostrophe gehören nicht zur Zeichenkette, mit einer Ausnahme: zwei aufeinanderfolgende Apostrophe werden als ein Apostroph der Zeichenkette aufgefaßt.

Die folgenden alphanumerischen Literale werden wie die entsprechenden hexadezimalen Werte behandelt. (Großbuchstaben beginnen mit der Codierung bei hexadezimal 41; Kleinbuchstaben beginnen bei 61; ein Leerzeichen hat die Codierung 20 und ein Apostroph ist als hexadezimal 27 codiert.)

Beispiele für alphanumerische Literale:

Eingabezeichen	hexadezimaler Wert
'A'	0041
'AB'	4142
'aA'	6141
''''	0027
''''''	2727
' A'	2041
'A '	4120

2.4. Registerwerte

In Ausdrücken können auch Register angegeben werden. Der Wert entspricht dann dem Inhalt des Registers im ZVE-Status des zu testenden Programms. Der Name des Registers steht für einen Wert. Wenn zum Beispiel an einem gewissen Punkt im Programm das BX-Register auf einen Datenbereich zeigt, der ausgegeben werden soll, dann zeigt das Kommando

```
DBX
```

den gewünschten Speicherbereich an.
Es ist zu beachten, daß bei der Eingabe von Assemblerbefehlen im A-Kommando die Registernamen anders behandelt werden als in Ausdrücken zu Kommandos. Bei Assemblerbefehlen nach A-Kommandos bezieht sich ein Registername auf ein Register, nicht auf seinen Inhalt.

2.5. Stack-Bezüge

Elemente des Stack können in Ausdrücken enthalten sein. Das Zeichen ^ bezeichnet einen 16-bit-Wert auf dem Stack-Top (adressiert durch das SS- und SP-Register im ZVE-Status des Nutzerprogramms). Eine Folge von n (^) bezeichnet den n-ten Wert des Stack.

Diese Besonderheit kann benutzt werden, um Unterbrechungspunkte auf die Rückkehr von einem Unterprogramm zu setzen, wenn nur bekannt ist, daß die Rückkehradresse auf dem Stack ist.
Die Kommandos

```
G,^
G,^^:^
```

setzen die Unterbrechungspunkte auf die Rückkehr von Subroutinen (near und far).

2.6. Symbolische Bezüge

SID86 ermöglicht den symbolischen Bezug auf Werte. Voraussetzung ist, daß die Symboltabelle zum Testprogramm geladen wurde. Ein symbolischer Bezug kann durch eine der folgenden Formen erfolgen:

```
(a) .s
(b) @s
(c) =s
```

wobei s eine Zeichenfolge repräsentiert, die einem Symbol in der Tabelle entspricht.

Die Form (a) erzeugt einen 16-bit-Wert, der dem Symbol entspricht, d. h. dem Wert des Symbols in der Symboltabelle.
Die Form (b) erzeugt den 16-bit-Wert, der in den zwei Speicher-Bytes enthalten ist, die durch das Symbol s spezifiziert werden.
Die Form (c) erzeugt einen 8-bit-Wert aus dem durch s spezifizierten Speicher-Byte. Die Formen (b) und (c) benutzen den Inhalt des DS-Registers als Segmentwert für die Speicheradresse.

Angenommen, die eingegebene Symboltabelle enthält die zwei Symbole:

```
0100 GAMMA   0102 DELTA
```

und der Speicher enthält ab Adresse 0100 im Data-Segment (spezifiziert durch DS) die folgenden Byte-Werte:

```
0100: 02
0101: 3E
0102: 4D
0103: 22
```

dann erzeugen die im folgenden dargestellten symbolischen Bezüge die entsprechenden Werte der rechten Spalte.

Symbolischer Bezug	Hexadezimaler Wert
.GAMMA	0100
.DELTA	0102
@GAMMA	3E02
@DELTA	224D
=GAMMA	0002
=DELTA	004D

2.7. Ausgewählte Symbole

In der Symboltabelle können Symbole mehrfach auftreten, wenn verschiedene Module eines Programms unabhängig voneinander die gleichen Namen für Daten oder Subroutinen verwenden. Daher ermöglicht SID86 den Bezug auf ausgewählte Symbole.
Die Form ist:

S1/S2/ ... /Sn

wobei S1 bis Sn Symbole in der Symboltabelle darstellen.
SID86 durchsucht die Symboltabelle vom ersten bis zum letzten Symbol in der Reihenfolge, in der die Symbole in der Datei auftreten. Für ein ausgewähltes Symbol sucht SID86 nach dem Symbol S1, dann weiter nach S2 bis das Symbol Sn gefunden wurde. Wenn dieser Suchvorgang nicht erfolgreich war, gibt SID86 das Fragezeichen (?) auf dem Bildschirm aus.
Hat beispielsweise die Symboltabelle in der Symboldatei die folgende Form:

0100 A 0300 B 0200 A 3E00 C 20F0 A 0102 A

und die Initialisierung des Datenbereiches sei wie im vorigen Beispiel, dann haben die folgenden Bezüge zu ausgewählten Symbolen die entsprechenden hexadezimalen Werte:

Symbolischer Bezug	Hexadezimaler Wert
.A	0100
@A	2E02
.A/A	0200
.C/A/A	0102
=C/A/A	004D
.B/A/A	20F0

2.8. Operatoren in Ausdrücken

Literalwerte, Zeichenketten und symbolische Bezüge können durch Operatoren zu symbolischen Ausdrücken verbunden werden. Die Operatoren sind einstelliges oder binäres "+" und "-". Die ganze Folge von Zahlen, Symbolen und Operatoren muß hintereinander ohne Leerzeichen eingegeben werden. Der Ausdruck wird von links nach rechts abgearbeitet. Durch jede Operation des Ausdrucks wird ein vier Zeichen langer hexadezimaler Wert gebildet. Überlauf und Unterlauf werden bei der Berechnung ignoriert. Der zuletzt berechnete Wert wird Kommandoparameter. Die Interpretation ist abhängig vom einzelnen Kommando.

Der Operator "+" zwischen zwei Operanden zeigt die Addition des rechten Operanden zu dem bis dahin gebildeten Wert an. Die Summe wird der neue gebildete Wert an dieser Stelle bei der Berechnung.

Der Operator "-" bewirkt die Subtraktion des rechten Operanden vom 16-bit-Wert, der bis dahin berechnet wurde. Wenn der Ausdruck mit Minus beginnt, dann wird der anfangs berechnete Wert als Null angenommen, d. h. -x wird berechnet wie 0-x.
Zum Beispiel ist das Kommando

```
DFF00-200,-#512
```

gleich dem einfachen Kommando

```
DFD00,FE00.
```

Zu Kommandos, die einen Adreßbereich spezifizieren (B, D, L, F, M und W), kann die Endadresse als Offset von der Startadresse spezifiziert werden. Dazu wird die Endadresse durch +Offset angegeben.
Zum Beispiel zeigt das Kommando

```
D121,+7
```

den Speicherbereich von 121 bis 128 (121+7) an. Die Verwendung des einstelligen Plus in einem anderen Zusammenhang ist nicht zulässig.

2.9. Beispiele für symbolische Ausdrücke

Die Schreibweise von symbolischen Ausdrücken in SID86 steht meist ganz in Beziehung zur Struktur des Programms, das getestet wird. Es soll beispielsweise ein Sortierprogramm mit den folgenden Größen getestet werden:

LIST: Basis einer Tabelle von Bytes, die sortiert werden sollen. Es sollen nicht mehr als 255 Elemente sein, also LIST(0), LIST(1), ... , LIST(254).

N: Byte-Variable, welche die aktuelle Anzahl von Werten in LIST enthält, wobei der Wert von N kleiner als 256 ist. Die zu sortierenden Elemente sind in LIST(0) bis LIST(N-1).

I: Byte-Index, der auf das nächste Element zeigt, das im Sortierprozess verglichen werden soll. LIST(I) ist das nächste Element, das in der Sortierfolge belegt wird, wobei I im Bereich von 0 bis N-1 liegt.

In diesem Datenfeld wird das Kommando

```
D.LIST,+#254
```

den ganzen Speicherbereich des Feldes, also

```
LIST(0), LIST(1), ... , LIST(254)
```

ausgeben. Das Kommando

```
D.LIST,+=I
```

zeigt den bereits sortierten LIST-Vektor einschließlich des nächsten Elementes, das sortiert werden soll:

```
LIST(0), LIST(1), ... , LIST(I)
```

Das Kommando

```
D.LIST+=I,+0
```

zeigt nur das Element LIST(I). Schließlich gibt das Kommando

```
D.LIST,+=N-1
```

nur den Bereich von LIST aus, der die aktuellen Elemente enthält, die sortiert werden sollen:

```
LIST(0), LIST(1), ... , LIST(N-1).
```

3. SID86-Kommandos

In diesem Abschnitt werden die Kommandos von SID86 und ihre Argumente beschrieben. SID86-Kommandos ermöglichen die Steuerung der Programmabarbeitung und erlauben die Anzeige und Änderung von Speicherinhalt und ZVE-Status.

3.1. A-Kommando (Assemble)

Das A-Kommando assembliert Mnemoniks der Assemblersprache direkt in den Speicher. Die Form ist:

As

wobei s die 20-bit-Adresse ist, wo die Assemblierung beginnen soll. SID86 antwortet auf das A-Kommando mit der Ausgabe der Adresse des Speicherplatzes, wo die Assemblierung beginnt. Nun können Assembleranweisungen eingegeben werden. Die Syntax der Anweisungen ist im Abschnitt 5. beschrieben. Nach der Eingabe konvertiert SID86 die Anweisung in die Binärform, speichert sie auf die Speicherplätze und gibt die nächste freie Adresse aus. Das wird solange fortgesetzt, bis der Bediener eine Leerzeile oder einen Punkt eingibt.

SID86 antwortet auf eine fehlerhafte Anweisung mit einem Fragezeichen (?) und wiederholt die Ausgabe der Adresse.

In Assembleranweisungen können immer dort, wo numerische Werte zulässig sind, auch symbolische Ausdrücke stehen. Es gibt einen Unterschied zwischen Ausdrücken in Assembleranweisungen und solchen, die sonst in SID86-Kommandos auftreten können. In einem A-Kommando bezieht sich die Angabe eines Registers auf den Namen des Registers, während in anderen Kommandos auf den Inhalt eines Registers Bezug genommen wird. In einem A-Kommando besteht keine Möglichkeit, durch einen Ausdruck auf den Inhalt eines Registers Bezug zu nehmen.
Im folgenden sind einige Beispiele von A-Kommandos angegeben.

```
#A213
```
Assemblieren ab Offset 213.

```
nnnn:0213 MOV AX,#128
```
Setzen des AX-Registers auf dezimal 128.

```
nnnn:0216 PUSH AX
```
Kellern des AX-Registers in den Stack.

```
nnnn:0217 CALL .PROC1
```
Aufruf einer Subroutine, deren Adresse dem Wert von PROC1 entspricht.

```
nnnn:021A TEST BYTE [.I/I],80
```
Test des höchswertigen Bits des Bytes, dessen Adresse der Wert des zweiten Symbols I ist.

```
nnnn:021E JZ .DONE
```
Sprung, falls das Zero-Flag gesetzt ist, zu dem Speicherplatz, dessen Adresse der Wert des Symbols DONE ist.

nnnn:0220 MOV AL,[.ARRAY+4]
Transport des Inhalts des Speicherbytes in das AL-Register, dessen Adresse der Wert des Symbols ARRAY+4 ist.

3.2. B-Kommando (Block Compare)

Das B-Kommando vergleicht zwei Speicherblöcke und gibt die Unterschiede auf dem Bildschirm aus. Die Form ist:

Bs1,f1,s2

Dabei ist s1 die 20-bit-Adresse des Beginns des ersten Speicherblocks; f1 ist der Offset des letzten Bytes des ersten Blocks; s2 ist die 20-bit-Adresse des Beginns des zweiten Speicherblocks. Wenn in s2 kein Segment spezifiziert ist, wird der Segmentwert von s1 benutzt.
Jede Differenz in den zwei Speicherbereichen wird auf dem Bildschirm in der Form

a1 b1 a2 b2

ausgegeben, wobei a1 und a2 die Adressen und b1 und b2 die Werte der Adressen sind. Werden keine Differenzen ausgegeben, sind die Blöcke identisch.

Beispiele für das B-Kommando:

#B40:0,1FF,60:0
Vergleich von 200H Bytes, beginnend ab 40:0 mit dem Block ab 60:0.

#BES:.ARRAY1,+FF,.ARRAY2
Vergleich von 256 Bytes, beginnend ab Offset ARRAY1 im Extra-Segment mit ARRAY2 im Extra-Segment.

3.3. D-Kommando (Display)

Das D-Kommando zeigt den Inhalt des Speichers als 8-bit- oder 16-bit-Hexadezimalwert und als KOI7-Zeichen an. Die Kommandoformen sind:

(a) D
(b) Ds
(c) Ds,f
(d) DW
(e) DWs
(f) DWs,f

wobei s eine 20-bit-Adresse ist, bei der die Anzeige beginnt, und f ist der 16-bit-Offset im spezifizierten Segment, wo die Anzeige beendet wird.

Der Speicherinhalt wird in einer oder mehreren Zeilen angezeigt. Jede Zeile zeigt bis zu 16 Werte. Für die ersten Kommandoformen erscheint die Anzeige wie folgt:

ssss:oooo bb bb ... bb cc ... c

Hierbei ist ssss der Segmentwert des Speicherbereiches und oooo der Offset innerhalb dieses Segments. Die Zeichen bb stellen den Speicherinhalt in hexadezimaler Form und die Zeichen c stellen den Speicherinhalt als KOI7-Zeichen dar. Ein Punkt steht bei der Ausgabe für nicht druckbare Zeichen.

Die Kommandoform (a) zeigt den Speicherinhalt ab der aktuellen Adresse in 12 Zeilen.
Die Form (b) ist gleich der Form (a), nur daß die Anfangsadresse der Anzeige zu Beginn auf s gestzt wird.
Kommandoform (c) zeigt den Speicherinhalt im Bereich von s und f.
Die nächsten drei Kommandoformen sind analog zu den ersten drei, nur daß die Speicherinhalte als 16-bit-Werte ausgegeben werden.
Die Ausgabeform ist:

```
ssss:bbbb wwww wwww ... www cccc ... cc
```

Während einer langen Anzeige kann das D-Kommando durch die Eingabe eines beliebigen Zeichens auf der Konsole abgebrochen werden.

Im folgenden sind einige Beispiele für das D-Kommando angegeben.

```
#DF00,F23
```
Zeigt die Bytes des Speichers von Offset F00H bis F23H im aktuellen Data-Segment.

```
#D.ARRAY+=I,+#10
```
Zeigt 11 Bytes, beginnend bei ARRAY(I) an.

```
#DW#128,#255
```
Zeigt den Inhalt des Speichers in Worten von 80H bis FFH.

3.4. E-Kommando (Load Program, Symbols for Execution)

Das E-Kommando lädt eine Datei in den Speicher, so daß folgende G-, T- oder U-Kommandos das Programm abarbeiten können. Ebenso können eine oder mehrere Symboldateien geladen werden.
Das E-Kommando hat die Formen:

```
(a) Efilespec
(b) Efilespec symfile, ..
(c) E*symfile, ...
(d) E
```

Die Form (a) lädt die durch filespec spezifizierte Datei mit Hilfe der BDOS-Ladefunktion. Es können nur Dateien im CMD-Dateiformat geladen werden. Falls kein Dateityp angegeben wurde, wird CMD angenommen. Der Inhalt der Segmentregister CS, DS, ES, SS und der Befehlszähler werden entsprechend der Informationen im Header der CMD-Datei gesetzt. Wenn die Datei vollständig geladen wurde, zeigt SID86 die Start- und Endadresse jedes geladenen Segments an. Das V-Kommando zeigt diese Informationen später während des Tests wiederholt an.

Die Form (b) lädt die Datei filespec wie die Kommandoform (a), zusätzlich werden eine oder mehrere Symboldateien geladen. Für die Symboldateien (symfile) wird der Dateityp SYM angenommem. Vor Beginn des Ladens der Symboldateien gibt SID86 die Mitteilung

aus. Falls SID86 in der Symboldatei Fehler erkennt, z. B. eine unzulässige hexadezimale Ziffer oder ein unzulässiges Symbol, dann erfolgt eine Fehlermitteilung und das Laden der Symboldatei wird abgebrochen. Die Symbole, die bis dahin geladen wurden, können mit dem H-Kommando ausgegeben werden. So läßt sich der Fehler in der SYM-Datei leicht ermitteln. Für die Symboldatei stehen maximal 64K Bytes zur Verfügung.

Falls mehrere Symboldateien zu einem Programm geladen werden, so müssen sie alle mit einem E-Kommando der Form (b) oder (c) geladen werden.
Die Kommandoform (c) lädt kein Programm sondern nur die spezifizierte Symboldatei. Die Kommandoform (d) gibt alle Speicherbereiche von vorher geladenen Programmen frei. Es wird keine Datei geladen.

Wenn eine Programmdatei mit dem E-Kommando geladen wird, so gibt SID86 alle Speicherbereiche von bereits geladenen Programmen (durch ein E- oder R-Kommando) frei. So kann jeweils nur ein Programm zur Abarbeitung geladen werden. Diese Programmdatei muß geladen werden, bevor die zugehörigen Symboldateien gelesen werden.

SID86 gibt eine Fehlermitteilung aus, falls die Datei nicht existiert oder nicht erfolgreich geladen werden kann.

Das Format der Symboldatei des Assemblers ist:

```
nnnn  symbol1   nnnn  symbol2 ...
  .
  .
  .
```

Dabei ist nnnn eine vier Zeichen lange hexadezimale Zahl. Symbolnamen können 31 Zeichen lang sein.
Im folgenden sind Beispiele des E-Kommandos angegeben:

```
#ETEST
```
Laden der Datei TEST.CMD

```
#ETEST.CMD TEST.SYM
```
Laden der Datei TEST.CMD und der Symboldatei TEST.SYM.

```
#ETEST TEST IO FORMAT
```
Laden der Programmdatei TEST.CMD und der Symboldateien TEST.SYM, IO.SYM und FORMAT.SYM.

```
#E* TEST1
```
Laden der Symboldatei TEST1.SYM.

3.5. F-Kommando (Fill)

Das F-Kommando füllt einen Speicherbereich mit einer Byte- oder Wortkonstanten. Die Kommandoformen sind:

(a) Fs,f,b
(b) FWs,f,w

wobei s die 20-bit-Startadresse des Blocks ist, der gefüllt werden soll, und f ist der 16-bit-Offset des letzten Bytes im Block innerhalb des durch s spezifizierten Segmentes.
Durch die Kommandoform (a) speichert SID86 den 8-bit-Wert b in den Bereich von s bis f.

In der Form (b) wird der 16-bit-Wert auf die Speicherplätze von s bis f in der Standardform gespeichert, d. h. zuerst die niederwertigen, dann die höherwertigen 8 Bit.
Falls s größer als f ist, oder der Wert von b größer als 255 ist, antwortet SID86 mit einem Fragezeichen. Wenn der gespeicherte Wert nicht richtig zurückgelesen werden kann, gibt SID86 eine Fehlermeldung aus. Das ist ein Zeichen für einen fehlerhaften oder nicht existierenden RAM auf der angegebenen Adresse.

Einige Beispiele für F-Komandos:

#F100,13F,0
Füllen des Speichers von 100H bis 13FH im laufenden Data-Segment mit dem Wert 0.

#F.ARRAY,+255,FF
Füllen eines Blocks von 256 Byte, beginnend ab ARRAY mit der Konstanten FFH.

3.6. G-Kommando (Go)

Das G-Kommando überträgt die Steuerung an das Programm, das getestet werden soll. Zusätzlich können ein oder zwei Unterbrechungspunkte gesetzt werden. Die Kommandoformen sind:

(a) G
(b) G,b1
(c) G,b1,b2
(d) Gs
(e) Gs,b1
(f) Gs,b1,b2
(g) -G
(mit den Formen a bis f)

Hier ist s eine 20-bit-Adresse, von der die Programmausführung zu starten ist, b1 und b2 sind 20-bit-Adressen der Unterbrechungspunkte. Falls für keine dieser Adressen ein Segmentwert spezifiziert wurde, wird standardmäßig der Inhalt des CS-Registers als Segmentwert benutzt.

In den Formen (a), (b) und (c) ist keine Startadresse spezifiziert, so wird die 20-bit-Adresse aus dem Inhalt der CS- und IP-Register gebildet. Die Form (a) überträgt die Steuerung an das Testprogramm, ohne einen Unterbrechungspunkt zu setzen.
Die Formen (b) und (c) setzen einen bzw. zwei Unterbrechungspunkte, bevor die Steuerung an das Testprogramm übertragen wird. Die

Formen (d), (e) und (f) sind analog zu den ersten drei Formen, nur daß die Register CS und IP zuerst auf s gesetzt werden.
Die Formen von (g) sind analog den Formen (a) bis (f), allerdings wird die Ausgabe von durchlaufenen Protokollierpunkten unterdrückt.

Nachdem SID86 die Steuerung an das Testprogramm übertragen hat, wird dieses unter Echtzeitbedingungen abgearbeitet, bis ein Unterbrechungspunkt erreicht wurde. An diesem Punkt erhält SID86 die Steuerung zurück, löscht alle Unterbrechungspunkte vom G-Kommando und zeigt die Adresse des erreichten Unterbrechungspunktes an:

```
*ssss:oooo   .symbol
```

Dabei entspricht ssss dem Inhalt des CS-Registers und oooo dem Inhalt des IP-Registers, bei dem die Unterbrechung erreicht wurde. Falls ein Symbol existiert, dessen Wert an dieser Stelle gleich oooo ist, wird es ausgegeben. Wenn ein Unterbrechungspunkt erreicht wird und SID86 die Steuerung zurückerhält, so ist der Befehl auf dieser Adresse noch nicht ausgeführt.

Nachfolgend einige Beispiele für G-Kommandos:

#G
Beginn der Programmausführung beim Stand der CS- und IP-Register ohne Unterbrechungspunkt.

#G.START,.ERROR
Beginn der Programmausführung bei der Marke START im Code-Segment, wobei ein Unterbrechungspunkt auf die Marke ERROR gesetzt wird.

#G,.ERROR,^
Fortsetzen der Programmausführung entsprechend CS- und IP-Register mit Unterbrechungspunkten auf der Marke ERROR und auf der Adresse im obersten Stack-Element. Das ist die Rückkehradresse der Routine, die getestet wird.

#-G,34F
Fortsetzen der Programmausführung mit einem Unterbrechungspunkt auf 34FH. Die Anzeige der Protokollierpunkte wird unterdrückt.

3.7. H-Kommando (Hexadecimal Math)

Das H-Kommando liefert verschiedene arithmetische Funktionen. Die Formen sind:

(a) Ha,b
(b) Ha
(c) H

Die Form (a) berechnet die Summe (ssss), die Differenz (dddd), das Produkt (pppppppp) und den Quotienten (qqqq) mit dem Rest (rrrr) zweier 16-bit-Werte. Die Ergebnisse werden in der Form

```
+ssss -dddd *pppppppp /qqqq (rrrr)
```

angezeigt. Bei Addition und Subtraktion werden Überlauf und Unterlauf ignoriert.

Die Form (b) zeigt den Wert des Ausdruckes a in hexadezimaler Form, als KOI7-Zeichen (falls der Wert des Ausdruckes alphanumerischen Zeichen entspricht) und in symbolischer Form (falls der Wert des Ausdruckes gleich dem Wert eines Symbols ist). Die Anzeige hat die Form:

```
hhhh #dddd 'c' .s
```

Die Form (c) zeigt die Symbole der aktuellen Symboltabelle. Jedes Symbol wird in der Form

```
nnnn  symbol-name
```

angezeigt. Die Anzeige kann durch Drücken einer beliebigen Taste auf der Konsole unterbrochen werden.

Im folgenden sind einige Beispiele von H-Kommandos aufgelistet:

#H
: Ausgabe aller Symbole und ihrer Werte, die mit dem E-Kommando geladen wurden.

#H.OPEN
: Zeigt den Wert des Symbols OPEN in hexadezimaler und dezimaler Form an.

#H@INDEX
: Zeigt den Inhalt eines Speicherwortes auf INDEX in hexadezimaler und dezimaler Form an.

#H5C28,80
: Anzeige von Summe, Differenz, Produkt und Quotient von 5C28H und 80H.

3.8. I-Kommando (Input Command Tail)

Das I-Kommando erstellt einen Dateisteuerblock und einen Kommandoparameter-Puffer in der Basisseite von SID86 und kopiert diese Informationen in die Basisseite der mit dem E-Kommando geladenen Datei. Das Kommando hat die Form:

```
Icommand-tail
```

Kommandoparameter (comand-tail) ist eine Zeichenkette, die gewöhnlich einen oder mehrere Dateinamen enthält. Der erste Dateiname wird in den Dateisteuerblock (FCB) ab 005CH eingetragen. Falls noch ein weiterer Dateiname als Kommandoparameter folgt, wird er ab 006CH eingetragen. Die gesamte Zeichenkette der Kommandoparameter wird außerdem in den Standard-Kommandopuffer ab 0080H eingetragen. Die Länge der Zeichenkette steht auf 80H, dann folgt die Zeichenkette. Sie wird mit einem Null-Byte abgeschlossen.
Falls ein Programm mit dem E-Kommando geladen wurde, dann kopiert SID86 Dateisteuerblock und Kommandopuffer von der Basisseite des SID86 in die Basisseite des geladenen Programms. Die Position der Basisseite von SID86 kann aus dem 16-bit-Wert auf der Adresse 0:6 ermittelt werden. Die Basisseite des geladenen Programms be-

ginnt ab DS:0.

Beispiele für das I-Kommando:

#IFILE1.CMD
Füllen des Dateisteuerblockes ab 5CH mit dem Namen FILE1.CMD und füllen des Puffers ab 80H mit der Zeichenkette 'file1.cmd' im Data-Segment des zuletzt mit dem E-Kommando geladenen Programms.

#IA:FILE1 B:FILE2 C:FILE3 ¤PX
Füllen des Dateisteuerblockes ab 5CH und 6CH mit A:FILE1 und B:FILE2 und kopieren der gesamten Zeichenkette ab 80H.

3.9. L-Kommando (List)

Das L-Kommando listet den Inhalt des Speichers in der Assemblersprache aus. Die Kommandoformen sind:

(a) L
(b) Ls
(c) Ls,f
(d) -L
(e) -Ls
(f) -Ls,f

Dabei ist s eine 20-bit-Adresse (List-Adresse) bei der die Listenausgabe beginnen soll und f ist ein 16-bit-Offset innerhalb des durch s spezifizierten Segments, auf dem die Ausgabe zu beenden ist.
Jeder reassemblierte Befehl hat die Form:

label:
ssss:oooo prefix opcode operands .symbol = memory-value

Dabei ist label ein Symbol, dessen Wert gleich dem Offset oooo ist, falls solch ein Symbol existiert; prefix kann sein LOCK, REPEAT, segment-override; opcode ist das Mnemonik für den Befehl; operands sind 0, 1 oder 2 Operanden, je nach Befehl; symbol ist das Symbol, dessen Wert gleich dem numerischen Operanden ist, falls solch ein Operand vorhanden ist und das Symbol dazu existiert.
Falls der Befehl auf einen Speicherplatz zugreift, wird der Inhalt des Speicherplatzes durch memory-value je nach Befehlsart, als ein Byte, Wort oder Doppelwort angegeben.

Die Form (a) gibt 12 reassemblierte Befehle ab der aktuellen List-Adresse aus.
Die Form (b) setzt zuerst die List-Adresse auf s und zeigt dann 12 Befehle an.
Form (c) reassembliert die Befehle von s bis f. Die lezten drei Formen sind analog den ersten drei, nur daß keine symbolischen Informationen (Marken und Symbole) ausgegeben werden.

In allen Fällen wird die List-Adresse zur Vorbereitung auf ein folgendes L-Kommando auf den nächsten Befehl gestellt. Wenn SID86 die Steuerung von einem Testprogramm erhält (nach einem G-, T- oder U-Kommando) dann ist die List-Adresse auf den aktuellen Wert des CS- und IP-Registers gesetzt.

Lange Anzeigen können durch das Drücken einer beliebigen Taste abgebrochen werden. Durch CTRL/S kann die Anzeige angehalten werden.
Die Syntax der Assembleranweisungen, wie sie vom L-Kommando erzeugt werden, ist im Abschnitt 5 beschrieben.

Falls auf der angegebenen Speicheradresse kein zulässiger Befehl gefunden wird, gibt SID86

```
??=nn
```

aus, wobei nn der hexadezimale Wert des Speicherplatzes ist.

Beispiele von L-Kommandos:

```
#L243C,244E
```
Reassemblieren der Befehle von 243CH bis 244EH.

```
#L.FIND,+20
```
Reassemblieren von 20H Bytes ab Marke FIND.

```
#L.ERR+3
```
Reassemblieren von 12 Zeilen ab Marke ERR+3.

3.10. M-Kommando (Move)

Das M-Kommando speichert den Inhalt eines Speicherbereiches in einen anderen Bereich. Die Form ist:

```
Ms,f,d
```

wobei s die 20-bit-Startadresse des Speicherbereiches ist, der umgespeichert werden soll; f ist der Offset des lezten Bytes dieses Speicherbereiches; d ist die 20-bit-Adresse des ersten Bytes des Bereiches, der die Daten erhalten soll. Falls in d kein Segment spezifiziert ist, wird der Segmentwert von s benutzt.
Falls d im Bereich zwischen s und f liegt, wird ein Teil des Speicherbereiches, der umgespeichert werden soll, überschrieben.

Beispiele für das M-Kommando:

```
#M20:2400,+9,30:100
```
Umspeichern von 10 Bytes von 20:2400 nach 30:100.

```
#M.ARRAY,+#63,.ARRAY2
```
Umspeichern von 64 Bytes von ARRAY nach ARRAY2.

3.11. P-Kommando (Pass-Point)

Das P-Kommando setzt, löscht und zeigt Protokollierpunkte an.
Die Formen sind:

```
(a) Pa,n
(b) Pa
(c) -Pa
(d) -P
(e) P
```

Ein Protokollierpunkt ist ein permanenter Unterbrechungspunkt. Er bleibt solange gesetzt, bis er explizit gelöscht wird. Im Ge-

gensatz dazu müssen Unterbrechungspunkte mit jedem G-Kommando neu gesetzt werden. Protokollierpunkte haben eine zugehörige Durchlaufzählung im Bereich von 1 bis 0FFFFH. Die Durchlaufzählung zeigt an, wie oft der Befehl auf dem Protokollierpunkt ausgeführt wird, bevor die Steuerung an die Konsole zurückgegeben wird. Es können bis zu 16 Protokollierpunkte gleichzeitig gesetzt werden.

Ein bedeutender Unterschied bei der Ausführung von Unterbrechungspunkten und Protokollierpunkten liegt darin, daß beim Erreichen eines Unterbrechungspunktes der Befehl auf dem Unterbrechungspunkt noch nicht ausgeführt ist, während bei der Unterbrechung durch einen Protokollierpunkt, also wenn der Durchlaufzähler 1 wird, der Befehl auf dem Protokollierpunkt ausgeführt wurde. Dadurch kann die Programmausführung nach einem Protokollierpunkt einfach mit einem G-Kommando fortgesetzt werden, ohne daß der gleiche Protokollierpunkt nochmals durchlaufen wird.

Die Kommandoformen (a) und (b) werden benutzt, um Protokollierpunkte zu setzen. Die Form (a) setzt einen Protokollierpunkt auf die Adresse a mit einer Durchlaufzählung von n, wobei a die 20-bit-Adresse des Protokollierpunktes und n die Durchlaufzählung im Bereich von 0 bis 0FFFFH ist. Falls auf der angegebenen Adresse bereits ein Protokollierpunkt aktiv ist, wird nur die Durchlaufzählung auf n gestzt. Falls bereits 16 Protokollierpunkte gesetzt sind, antwortet SID86 mit einem Fragezeichen.

Die Form (b) setzt einen Protokollierpunkt auf die Adresse a mit einer Durchlaufzählung von 1. Falls auf a bereits ein Protokollierpunkt gesetzt ist, wird die Durchlaufzählung nur auf 1 gesetzt. SID86 antwortet mit einem Fragezeichen, falls bereits 16 Protokollierpunkte aktiv sind.

Die Formen (c) und (d) werden genutzt, um Protokollierpunkte zu löschen. Die Form (c) löscht den Protokollierpunkt auf a. Falls dort kein Protokollierpunkt gesetzt war, antwortet SID86 mit einem Fragezeichen. Die Form (d) löscht alle gesetzten Protokollierpunkte.
Die Form (e) zeigt alle aktiven Protokollierpunkte an. Die Form der Anzeige ist:

```
nnnn ssss:oooo .symbol
```

wobei nnnn der aktuelle Stand der Durchlaufzählung ist, ssss:oooo sind Segment und Offset der Adresse des Protokollierpunktes und .symbol ist der symbolische Name des Offsets des Protokollierpunktes, falls solch ein Symbol existiert.

Wenn ein Protokollierpunkt erreicht wird, gibt SID86 folgende Information aus:

```
nnnn PASS ssss:oooo .symbol
```

Die Bedeutung der Abkürzungen ist bereits oben beschrieben. Danach zeigt SID86 den ZVE-Zustand an, bevor der Befehl auf dem Protokollierpunkt abgearbeitet wird. SID86 führt dann den Befehl auf dem Protokollierpunkt aus. Wenn der Durchlaufzähler größer 1 ist, wird er um 1 verringert und die Steuerung geht an das Testprogramm zurück.
Wenn die Durchlaufzählung 1 erreicht, gibt SID86 die Unterbrechungsadresse aus (das ist die Adresse des nächsten Befehls).

*ssss:oooo .symbol

Wenn die Durchlaufzählung einmal 1 erreicht hat, so bleibt sie auf 1 bis der Protokollierpunkt gelöscht oder die Durchlaufzählung neu gesetzt wird.
Protokollierpunkte sind im Zusammenhang mit G-, T- oder U-Kommandos zu verwenden. Bei G- oder U-Kommandos kann die Anzeige des Durchlaufes durch einen Protokollierpunkt durch -G oder -U unterdrückt werden. In diesem Fall wird nur das Erreichen des Protokollierpunktes bei der Durchlaufzählung =1 angezeigt. Die Ausführung der G- oder U-Kommandos kann durch Drücken einer beliebigen Taste auf der Konsole jederzeit unterbrochen werden. SID86 bricht dann bei dem nächsten Protokollierpunkt ab und wartet auf die Eingabe eines Kommandos.

Protokollierpunkte können auch im Zusammenhang mit Unterbrechungspunkten von G-Kommandos verwendet werden. Falls ein Protokollierpunkt und ein Unterbrechungspunkt auf die gleiche Adresse gesetzt wurden, wird der Unterbrechungspunkt zuerst erreicht. Sonst verhalten sich Protokollierpunkte in gewohnter Weise. Der Durchlaufzähler wird um 1 verringert und wenn er 1 erreicht, geht die Steuerung wieder an SID86.
Standardmäßig werden die Segmentregister an Protokollierpunkten nicht angezeigt. Das S- bzw. -S-Kommando schaltet die Anzeige der Segmentregister ein bzw. aus. Siehe dazu die Beschreibung des S-Kommandos.

Beispiele zum P-Kommando:

#P
Zeigt die aktiven Protokollierpunkte an.

#P.ERROR
Setzt einen Protokollierpunkt auf ERROR.

#P.PRINT,17
Setzt einen Protokollierpunkt auf PRINT und die Durchlaufzählung auf 17.

#-P
Löscht alle Protokollierpunkte.

#-P.ERROR
Löscht den Protokollierpunkt auf ERROR.

3.12. R-Kommando (Read)

Das R-Kommando liest eine Datei in einen zusammenhängenden Speicherbereich. Die Formen sind:

(a) Rfilespec
(b) Rfilespec,s

wobei filespec die Spezifikation der Datei ist, die gelesen werden soll und s ist der Speicherplatz, auf dem die Datei beginnen soll.
Es muß stets eine vollständige Dateispezifikation angegeben werden. Das R-Kommando nimmt keinen Standard-Dateityp an. Das R-Kommando liest eine Datei einschließlich Datei-Header in den Speicher und setzt CS- und DS-Register auf die Anfangsadresse des

belegten Speicherbereiches.

Das R-Kommando wird besonders im Zusammenhang mit dem W-Kommando benutzt.

In der Form (a) bestimmt SID86 die Anfangsadresse der Datei im Speicher.
Die Form (b) veranlaßt SID86, die Datei in den Speicher ab Segment s zu lesen. Diese Adresse wird in der Standardform (ssss:oooo) angegeben, zum Beispiel:

```
#RSCP.SYS,1000:0
```

Die niederwertigen vier Bits von s werden als Null angenommen, so daß SID86 die Dateien ab Paragraphgrenze liest. Falls der Speicher ab s nicht verfügbar ist, gibt SID86 die Fehlermitteilung

```
MEMORY REQUEST DENIED
```

aus. SID86 liest die spezifizierte Datei in den Speicher, berechnet die Anfangs- und Endadresse der Datei und zeigt diese auch an.

Durch ein V-Kommando können diese Informationen später wieder angezeigt werden. Der Standard-Zeiger für die Anzeige (für folgende D-Kommandos) wird auf die Anfangsadresse des Blockes gesetzt, den die Datei belegt.

Das R-Kommando gibt keinen Speicherbereich frei, der vorher durch R- oder E-Kommandos belegt wurde. Auf diese Weise können mehrere Dateien gleichzeitig geladen werden. Es können maximal sieben Dateien unter SID86 geladen werden, das ist die Anzahl von Speicherbereichen, die gleichzeitig unter BDOS verwaltet werden können, minus 1 für SID86 selbst.

SID86 gibt eine Fehlermitteilung aus, falls eine Datei nicht existiert oder wenn nicht genügend Speicher frei ist, um die Datei zu laden.

Beispiele von R-Kommandos:

```
#RSID86.CMD
```
Laden von SID86.CMD in den Speicher.

```
#RTEST.CMD
```
Laden der Datei TEST.CMD in den Speicher.

```
#RTEST.CMD,1000:0
```
Laden der Datei TEST.CMD, beginnend ab 1000:0.

3.13. S-Kommando (Set)

Das S-Kommando ändert den Inhalt von Bytes oder Worten im Speicher. Die Formen sind:

(a) Ss
(b) SWs
(c) S
(d) -S

wobei s die 20-bit-Speicheradresse ist, auf der die Änderung erfolgen soll.

SID86 gibt auf das Kommando die Speicheradresse und deren Inhalt auf der nächsten Zeile aus. Als Antwort auf Form (a) erscheint:

```
ssss:oooo bb
```

und als Antwort auf Form (b) wird

```
ssss:oooo wwww
```

ausgegeben, wobei bb und wwww der Speicherinhalt in Byte- bzw. Wortformat ist.

Als Antwort auf die Anzeige kann entweder der Speicherinhalt geändert werden oder er kann unverändert gelassen werden. Wird ein zulässiger Ausdruck eingegeben, so wird der Inhalt des Bytes bzw. Wortes durch den Wert des Ausdrucks ersetzt.
Wird kein Wert eingegeben, sondern nur mit (CR) geantwortet, bleibt der Speicherinhalt unverändert. In beiden Fällen setzt SID86 die Anzeige fort, gibt die nächste Speicheradresse und deren Inhalt aus, bis ein Punkt als Endekennzeichen eingegeben wird oder SID86 einen fehlerhaften Wert erkannt hat.

Als Antwort auf Form (a) kann auch eine Zeichenkette eingebenen werden, die mit Anführungszeichen (") beginnt und mit (CR) endet. Die Zeichen zwischen Anführungszeichen und (CR) werden in den Speicher ab der angezeigten Adresse eingetragen. Es erfolgt keine Konvertierung von Groß-/Kleinbuchstaben. Die nächste angezeigte Adresse ist dann die erste Adresse nach der Zeichenkette.

SID86 gibt eine Fehlermitteilung aus, falls der eingespeicherte Wert nicht korrekt wieder gelesen werden kann. Die Ursache kann ein fehlerhafter oder nicht existierender RAM auf dem angezeigten Speicherplatz sein.

Die Formen (c) und (d) steuern die Ausgabe der Segmentregister, wenn bei Trace-Kommandos oder an Protokollierpunkten der ZVE-Status angezeigt wird. Form (c) schaltet die Anzeige der Segmentregister ein und Form (d) schaltet sie wieder aus. Es ist meist vorteilhaft, die Anzeige der Segmentregister während des Tests auszuschalten, damit die Anzeige des ZVE-Status auf eine Zeile paßt.

Beispiele von S-Kommandos:

```
#S.ARRAY+3
```
Beginn der Eingabe bei ARRAY+3.

```
nnnn:1234 55  0
          Byte auf Null setzen.

nnnn:1235 55  "abc
          3 Bytes auf 'a', 'b', 'c' setzen.

nnnn:1238 55  #75
          Byte auf dezimal 75 setzen.

nnnn:1239 55
          Beenden des S-Kommandos.

#S
          Segmentregister-Anzeige bei der Anzeige des  ZVE-Status
          einschalten.

#-S
          Ausschalten der Segmentregister-Anzeige.
```

3.14. T-Kommando (Trace)

Das T-Kommando protokolliert die Abarbeitung des Testprogramms in 1 bis OFFFFH Programmschritten und zeigt den ZVE-Status vor Abarbeitung jedes Befehls (trace). Die Formen sind:

(a) T
(b) Tn
(c) TW
(d) TWn
(e) -T
(mit den Formen a bis d)

wobei n die Anzahl der Programmschritte ist, die ausgeführt werden sollen, bis die Steuerung an die Konsole zurückgegeben wird. Fehlt n, so wird nur ein einzelner Befehl ausgeführt.

Ein Programmschritt ist im allgemeinen ein einzelner Befehl mit den folgenden Ausnahmen:

- Falls ein BDOS-Ruf getestet wird, so behandelt SID86 die ganze BDOS-Funktion als einen Programmschritt, der im Direktlauf ausgeführt wird.

- Falls eine Befehlsfolge mit MOV oder POP getestet wird, wobei der Zieloperand ein Segmentregister ist, so führt die ZVE den nächsten Befehl unmittelbar aus. Der Prozessor verbietet Interrupts (einschließlich des Trace-Interrupts) für Befehle nach solchen Segmentregister-Ladeopertionen. Zum Beispiel kann die Befehlsfolge

  ```
  MOV SS,STACKSEGMENT
  MOV SP,STACKOFFSET
  ```

  bei der Ausführung nicht unterbrochen werden. Eine Folge solcher MOV- oder POP-Operationen plus eine Operation nach der Folge wird als ein Programmschritt behandelt.

- Falls eine der TW-Kommandoformen genutzt und mit CALL, CALLF oder INT getestet wird, so wird die gesamte Subroutine oder der Interrupt-Handler (und jeder Subroutine-Aufruf darin) als ein

Programmschritt behandelt und im Direktlauf abgearbeitet.

Bevor ein Programmschritt ausgeführt wird, zeigt SID86 folgende Informationen an:

- den Zustand der ZVE
- den reassemblierten Befehl, der als nächstes abgearbeitet werden soll
- symbolische Namen von Operanden (falls vorhanden)
- Inhalt der Speicherplätze, auf die durch die Operation zugegriffen wird (falls vorhanden)

Im Abschnitt 3.18. wird eine detaillierte Beschreibung der Anzeige des ZVE-Status für das X-Kommando gegeben. Wenn ein Symbol existiert, dessen Wert gleich dem Inhalt des IP-Registers ist, so wird erst der Symbolname, gefolgt von einem Doppelpunkt und dann die Zeile mit dem ZVE-Status ausgegeben.
Die Segmentregister werden beim T-Kommando nicht angezeigt. Dadurch kann der gesamte ZVE-Status auf einer Zeile ausgegeben werden. Wird die Segmentregister-Anzeige gewünscht, muß das S-Kommando benutzt werden (siehe Abschnitt 3.13.).

Bei allen Kommandoformen wird die Steuerung an das Testprogramm auf der Adresse übertragen, die durch die CS-Register und IP-Register angezeigt werden. Falls n nicht spezifiziert ist, wie in Form (a), wird ein Programmschritt ausgeführt.
Sonst führt SID86 n Programmschritte aus und zeigt den ZVE-Status vor Ausführung jedes Befehls an. Lange Protokolle können durch das Drücken einer beliebigen Taste auf der Konsole abgebrochen werden.

Wenn n Programmschritte ausgeführt wurden, zeigt SID86 die Adresse des nächsten Befehls an, der ausgeführt werden soll, zusammen mit dem symbolischen Wert des IP-Registers, falls ein solches Symbol existiert.

```
*ssss:oooo .symbol
```

Die Formen (c) und (d) sind analog zu den Formen (a) und (b), nur werden Subroutinen-Rufe anders behandelt. Bei der TW-Form wird die gesamte Subroutine, die vom aktuellen Programm-Niveau gerufen wird, als ein Programmschritt behandelt. Die Subroutine wird im Direktlauf abgearbeitet. Das ermöglicht ein komfortables Testen, weil Subroutinen, die bereits fehlerfrei oder aus anderen Gründen uninteressant sind, übergangen werden können.

Beginnt das T-Kommando mit einem Minuszeichen (-) wie in Form (e), so werden alle symbolischen Verweise der Anzeige unterdrückt. Dadurch kann die Geschwindigkeit der Anzeige erhöht werden, denn der Suchlauf in der Symboltabelle wird übergangen.

Wenn ein einzelner Befehl getestet wird, sind Interrupts für die Dauer des Befehls verboten. SID86 testet auch nicht durch Interrupt-Handler.

Nach einem T-Kommando wird die List-Adresse des nächsten auszuführenden Befehls gesetzt. Die Standardsegmentwerte werden auf die Werte der CS- und DS-Register gesetzt.

Beispiele von T-Kommandos:

#T
Einen Programmschritt testen.

#TFFFF
65535 Programmschritte testen.

#-T#500
500 Programmschritte testen ohne symbolische Ausgabe.

3.15. U-Kommando (Untrace)

Das U-Kommando ist dem T-Kommando ähnlich. Die Anzeige des ZVE-Status erfolgt nur einmal, vor der Ausführung des ersten Programmschrittes. Die Formen sind:

(a) **U**
(b) **Un**
(c) **UW**
(d) **UWn**
(e) **-U**
(mit den Formen a bis d)

wobei n die Anzahl der Befehle ist, die ausgeführt wird, bevor die Steuerung an die Konsole zurückgegeben wird. Das U-Kommando kann durch Drücken einer beliebigen Taste an der Konsole jederzeit abgebrochen werden, bevor n Programmschritte ausgeführt worden sind.

Die Formen (e) unterscheiden sich in der Wirkung von den analogen T-Kommandos. Die -U-Form unterdrückt die Anzeige von erreichten Protokollierpunkten. Nur wenn die Durchlaufzählung 1 erreicht, erfolgt eine Mitteilung.

Beispiele von U-Kommandos:

#U200
Ausführen von 200H Programmschritten ohne Anzeige der Befehle.

#-U200
Ausführen von 200H Programmschritten, Unterdrücken der Anzeige von Kontrollpunkten.

3.16. V-Kommando (Value)

Das V-Kommando zeigt Informationen über die zuletzt mit einem E- oder R-Kommando geladene Datei an. Symboltabellen werden nicht berücksichtigt. Die Form ist:

V

Wurde die letzte Datei mit einem E-Kommando geladen, wird die Start- und Endadresse jedes Segments der Datei angezeigt. Wurde die letzte Datei mit einem R-Kommando gelesen, dann zeigt das V-Kommando die Start- und Endadresse des Speicherblockes, wohin die Datei gelesen wurde. SID86 antwortet mit einem Fragezeichen, falls keine Datei geladen wurde.

3.17. W-Kommando (Write)

Das W-Kommando schreibt den Inhalt eines zusammenhängenden Speicherblockes auf die Platte. Die Formen sind:

(a) Wfilespec
(b) Wfilespec,s,f

wobei filespec die Dateispezifikation ist, welche auf die Platte geschrieben wird und die Daten enthält; s und f sind die 20-bit-Anfangs- und Endadresse des Speichblockes, der geschrieben werden soll. Falls in f kein Segment spezifiziert ist, wird der Wert von s benutzt.

Bei der Form (a) nimmt SID86 die Grenzen s und f von der Datei, die zuletzt mit einem R-Kommando gelesen wurde. Diese Form ist vorteilhaft zum Herstellen von Dateien nachdem Korrekturen durchgeführt wurden, ohne daß die Länge der Datei geändert wurde. SID86 antwortet mit einem Fragezeichen, falls keine Datei mit einem R-Kommando gelesen wurde.
Bei der Form (b) wird angenommen, daß die letzten vier Bits der Adresse s Null sind. Ein Speicherblock, der geschrieben werden soll, muß immer an einer Paragraphgrenze beginnen.

Falls die im W-Kommando spezifizierte Datei bereits existiert, löscht SID86 diese, bevor mit dem Schreiben der neuen Datei begonnen wird.

Beispiele von W-Kommandos:

#WTEST.CMD
Schreiben von TEST.CMD auf die Platte. Die Länge des Speicherblockes wird aus dem lezten R-Kommando ermittelt.

#WB:TEST.CMD,40:0,3FFF
Schreiben des Speicherblockes 40:0 bis 40:3FFF in die Datei TEST.CMD auf dem Laufwerk B.

3.18. X-Kommando (Examine CPU-Status)

Das X-Kommando ermöglicht es, den ZVE-Status des Testprogramms anzuzeigen und zu ändern. Die Formen sind:

(a) X
(b) Xr
(c) Xf

wobei r der Name eines ZVE-Registers und f die Abkürzung für ein Flag ist. Die Form (a) zeigt den ZVE-Status im Format:

```
           AX   BX  ...  SS   ES   IP
--------- xxxx xxxx ... xxxx xxxx xxxx
instruction .symbol-name = memory-value
```

Die neun Striche am Anfang der Zeile zeigen den Zustand der Flags an. Jede Position kann entweder ein Strich sein, zum Zeichen, daß das entsprechende Flag nicht gesetzt ist (0), oder eine Abkürzung aus einem Zeichen für den Flag-Namen, dann ist das Flag gesetzt (1). Die Abkürzungen für die Flag-Namen sind:

Zeichen	Flag
O	Overflow
D	Direction
I	Interrupt Enable
T	Trap
S	Sign
Z	Zero
A	Auxilary Carry
P	Parity
C	Carry

Auf der folgenden Zeile ist instruction der reassemblierte Befehl auf dem nächsten Speicherplatz, der ausgeführt werden soll und dessen Positon durch die Register CS und IP bestimmt ist. Falls die Symboltabelle ein Symbol enthält, dessen Wert gleich dem eines Operanden in instruction ist, so wird der Symbolname im Feld symbol-name angezeigt. Er beginnt mit einem Punkt. Falls der Befehl zu Speicherplätzen zugreift, so wird der Inhalt dieser Adresse im Feld memory-value angezeigt. Die Werte beginnen mit einem Gleichheitszeichen. In Abhängigkeit vom Befehl wird entweder ein Byte, ein Wort oder ein Doppelwort angezeigt. Zusätzlich zur Anzeige des ZVE-Status setzt die Form (a) die Werte der Standardsegmente zurück auf die CS- und DS-Registerwerte und den Standard-Offset für das L-Kommando auf den Wert des IP-Registers.

Die Form (b) ermöglicht es, die Registerwerte im ZVE-Status des Testprogramms zu ändern. Das r nach dem X ist der Name der 16-bit-ZVE-Register. SID86 antwortet auf das Kommando mit Namen und Inhalt des entsprechenden Registers. Wird (CR) eingegeben, bleibt der Inhalt der Register unverändert. Wird ein zulässiger Ausdruck eingegeben, so wird der Inhalt des Registers auf den Wert des Ausdruckes gesetzt. In beiden Fällen wird das nächste Register angezeigt. Das kann solange fortgesetzt werden, bis ein Punkt oder ein unzulässiger Ausdruck eingegeben wird oder das letzte Register angezeigt wurde.

Die Form (c) ermöglicht es, eines der Flags im ZVE-Status des Testprogramms zu setzen. SID86 antwortet auf das Kommando mit dem Namen des Flags und dessen Zustand. Falls ein (CR) eingegeben wird, bleibt der Zustand des Flags unverändert. Wird ein zulässiger Wert eingegeben, so wird das Flag in den Zustand versetzt. Zulässige Werte sind nur 0 und 1. Mit dem Xf-Kommando kann jeweils nur ein Flag angezeigt oder verändert werden.

Beispiele von X-Kommandos:

```
#XBP
          Ändern der Registerinhalte, beginnend mit BP.

BP=1000   2B64
          Ändern von BP auf hexadezimal 2B64.

SI=2000   #12345
          Ändern von SI auf dezimal 12345.

DI=0020   .VAR+6
          Ändern von DI auf den Wert des Symbols VAR+6.

CS=0040   .
          Beenden des X-Kommandos.
```

4. Standardsegmentwerte

SID86 hat einen internen Mechanismus, der die aktuellen Segmentwerte aufbewahrt. Diese werden für die wahlweise Segmentspezifikation in den Kommandos von SID86 benötigt. SID86 unterteilt seinen Kommandosatz in Abhängigkeit vom Segment in zwei Kommandotypen, der standardmäßig verwendet wird, falls keine Segmentspezifikation im Kommando angegeben ist.

Der erste Kommandotyp bezieht sich auf das Code-Segment. Dazu gehören das A-, L-, P- und W-Kommando. Diese Kommandos benutzen den internen Typ1-Segmentwert, falls im Kommando kein Segmentwert angegeben wurde.

Beim Aufruf von SID86 wird der Typ1-Segmentwert auf 0 gesetzt. Folgende Kommandos ändern seinen Wert:

- Wenn ein E-Kommando eine Datei lädt, so setzt SID86 den Typ1-Segmentwert auf den Wert des CS-Registers.

- Wenn ein R-Kommando eine Datei lädt, dann setzt SID86 den Typ1-Segmentwert auf das Basissegment, wohin die Datei gelesen wurde.

- Wenn ein X-Kommando den Wert des CS-Registers ändert, so setzt SID86 den Typ1-Segmentwert auf den neuen Wert des CS-Registers.

- Wenn SID86 die Steuerung von einem Testprogramm nach einem G-, T- oder U-Kommando zurückerhält, so setzt SID86 den Typ1-Segmentwert auf den Wert des CS-Registers.

- Wenn ein A- oder L-Kommando einen Segmentwert explizit spezifiziert, so setzt SID86 den Typ1-Segmentwert auf den angegebenen Segmentwert.

Die zweite Gruppe von Kommandos bezieht sich auf das Data-Segment. Dazu gehören das D-, F-, M- und S-Kommando. Diese Kommandos benutzen den internen Typ2-Segmentwert, falls im Kommando kein Segmentwert spezifiziert wurde.

Beim Aufruf von SID86 wird der Typ2-Segmentwert auf 0 gesetzt. Folgende Kommandos ändern seinen Wert:

- Wenn ein E-Kommando eine Datei lädt, so setzt SID86 den Typ2-Segmentwert auf den Wert des DS-Registers.

- Wenn ein R-Kommando eine Datei liest, so setzt SID86 den Typ2-Segmentwert auf den Wert des Basissegments, wohin die Datei gelesen wurde.

- Wenn ein X-Kommando den Wert des DS-Registers verändert, so setzt SID86 den Typ2-Segmentwert auf den neuen Wert des DS-Registers.

- Wenn SID86 die Steuerung von einem Testprogramm nach einem G-, T- oder U-Kommando zurückerhält, so setzt es den Typ2-Segmentwert auf den Wert des DS-Registers.

- Wenn in einem D-, F-, M- oder S-Kommando explizit ein Segmentwert spezifiziert wird, so setzt SID86 den Typ2-Segmentwert auf den angegebenen Segmentwert.

Wenn Programme abgearbeitet werden, die für die CS- und DS-Register identische Werte haben, dann haben alle SID86-Kommandos die gleichen Standardsegmentwerte, falls sie nicht explizit verändert werden.

Tabelle 2 enthält alle Kommandos von SID86 und die zugehörigen Standardsegmentwerte. Das G-Kommando hat keinen Standardsegmentwert, da es sich auf das CS-Register bezieht.

Tabelle 2: Kommandos und Standardsegmentwerte

Kommando	Typ1	Typ2
A	*	
B		*
D		*
E	@	@
F		*
G	@	@
H		
I		
L	*	
M		*
P	u	
R	*	@
S		*
T	@	@
U	@	@
V		
W	*	
X	@	@

* Benutzt den Standardsegmentwert, falls kein Segmentwert angegeben ist; ändert den Standardsegmentwert, falls ein Segmentwert spezifiziert ist.
@ Ändert diesen Standardsegmentwert.
u Benutzt diesen Standardsegmentwert, falls kein Segmentwert spezifiziert ist.

5. Assemblersprache für A- und L-Kommandos

Im allgemeinen ist die Syntax der Assembler-Anweisungen in den A- und L-Kommandos mit der Assemblersprache identisch. Es gibt nur die folgenden Ausnahmen:

- Bis zu drei Präfixe (LOCK, Repeat, Segment-Override) können in einer Anweisung auftreten. Sie müssen vor dem Operationscode der Anweisung stehen. Außerdem kann ein Präfix auch allein auf einer Zeile eingegeben werden.

- Ein Unterschied für Zeichenkettenbefehle für Byte- und Wortlänge wird gemacht.

Byte	Wort	Byte	Wort
LODSB	LODSW	MOVSB	MOBSW
STOSB	STOSW	CMPSB	CMPSW
SCASB	SCASW		

- Die Mnemoniks für Sprünge und Rufe sind:

kurz	normal	weit
JMPS	JMP	JMPF
	CALL	CALLF
	RET	RETF

- Wenn der Operand in CALLF oder JMPF eine absolute 20-bit-Adresse ist, so wird sie in der Form
  ssss:oooo
  eingegeben, wobei ssss der Segmentwert und oooo der Offset der Adresse ist.

- Mehrdeutige Operanden, die entweder zu Bytes oder Wörtern zugreifen können, müssen durch die Präfixe BYTE oder WORD spezifiziert werden. Diese Präfixe können auch durch BY oder WO abgekürzt werden. Zum Beispiel:

```
INC BYTE [BP]   oder INC BY [BP]
NOT WORD [1234] oder NOT WO [1234]
```

- Operanden zur direkten Speicheradressierung werden in eckige Klammern eingeschlossen, um sie von Direktwerten unterscheiden zu können.

```
ADD  AX,5      ; Addiere 5 zum Register AX
ADD  AX,[5]    ; Addiere den Inhalt von Speicherplatz 5 zu AX
```

- Die Formen der indirekten Registeroperanden sind:
  [pointer-register]
  [index-register]
  [pointer-register + index-register]
  wobei pointer-register die Register BX und BP und index-register die Register SI und DI sein können. Vor jedem dieser Ausdrücke kann noch ein numerischer Offset stehen.

Zum Beispiel:

```
ADD  BX,[BP+SI]      ADD  BX,1D47[BP+SI]
ADD  BX,3[BP+SI]
```

6. Beispiel für einen Dialog mit SID86

Im folgenden Beispeil testet der Nutzer interaktiv ein einfaches Programm. Das Programm und die Kommandos sind ausführlich kommentiert.

```
; Assemblieren des Programms SAMPLE.A86


B>ASM86 SAMPLE

ASM86 V 1.0
END OF PASS 1
END OF PASS 2
END OF ASSEMBLY.  NUMBER OF ERRORS:   0.  USE FACTOR:  0%


; Ausgabe der Quellprogramm-Liste

B>TYPE SAMPLE.A86
```

```
                        ; SID86 SAMPLE PROGRAM


   0000                 RESET   EQU     0       ; SYSTEM RESET
   0009                 PSTRING EQU     9       ; PRINT UNTIL ¤
   000A                 RSTRING EQU     10      ; READ CONSOLE BUFFER

   000D                 CR      EQU     0DH     ; CARRIAGE RETURN
   000A                 LF      EQU     0AH     ; LINE FEED
   0024                 DOLLAR  EQU     024H    ; DOLLAR

                        ;       CODE STARTS HERE

0000 BA0001             START:  MOV     DX,OFFSET MESSAGE
0003 E81200     0018            CALL    PRINT           ; PRINT MESSAGE

0006 BA1A01                     MOV     DX,OFFSET INPLINE
0009 E81200     001E            CALL    ACCEPT          ; ACCEPT INPUT LINE

000C E81D00     002C            CALL    BACKST          ; STORE BACKWARDS INPUT

000F BA5A01                     MOV     DX,OFFSET PRINTL
0012 E80300     0018            CALL    PRINT           ; PRINT STORED CHARACTERS

0015 E90C00     0024            JMP     EXIT            ; RETURN CONTROL

                        ;       SOUBROUTINES


                        ;       PRINT THE BUFFER ADDRESSED BY DX UNTIL ¤

                        PRINT:
0018 B109                       MOV     CL,PSTRING      ; FUNCTION CODE
001A E80C00     0029            CALL    BDOS            ; TYPE THE STRING
001D C3                         RET
```

```
ASM86 V 1.0     SOURCE: SAMPLE.A86
                     PAGE   2

                        ;       ACCEPT CONSOLE BUFFER

                        ACCEPT:
001E B10A                       MOV     CL,RSTRING      ; FUNCTION CODE
0020 E80600       0029          CALL    BDOS            ; ACCEPT THE LINE
0023 C3                         RET

                        ;       EXIT ROUTINE

                        EXIT
0024 B100                       MOV     CL,RESET        ; FUNCTION CODE
0026 E80000       0029          CALL    BDOS            ; RETURN

                        ;       BDOS ROUTINE

                        BDOS:
0029 CDE0                       INT     224             ; ENTRY TO BDOS
002B                            RET

                        ;       STORE BACKWARDS CHARACTERS, FILL PRINT LINE

                        BACKST:
002C BF0000                     MOV     DI,0            ; SET DEST INDEX
002F 8B361B01                   MOV     SI,WORD PTR CCOUNT ; GET NUMBER OF CHARS
0033 81E6FF00                   AND     SI,0FFH         ; SOURCE INDEX
                        STORE:
0037 8A841C01                   MOV     AL,INBUF[SI]    ; GET INPUT CHARACTER
003B 88855C01                   MOV     OUTBUF[DI],AL   ; MOV TO OUTPUT
003F 47                         INC     DI              ; INCREMENT DEST INDEX
0040 4E                         DEC     SI              ; DECREMENT SOURCE INDEX
0041 75F4         0037          JNZ     STORE           ; REPEAT IF SI NOT ZERO

0043 C6855C0124                 MOV     OUTBUF[DI],DOLLAR ; APPEND A ¤ FOR PRINT
004B C3                         RET
```

```
                                DSEG
                                ORG     100H            ; SPACE FOR BASE PAGE

0100 0D0A               MESSAGE DB      CR,LF

0102 534944383620               DB      'SID86 SAMPLE SESSION'
     53414D504C45
     205345535349
     4F4E
0116 0D0A0A24                   DB      CR,LF,LF,DOLLAR
011A 3E                 INPLINE DB      62          ; READ BUFFER STARTS WITH MAX LENGTH
011B                    CCOUNT  RB      1           ; NUMBER OF CHARS FROM BDOS
011C                    INBUF   RB      62          ; INPUT BUFFER
015A 0D0A               PRINTL  DB      CR,LF       ; PRINT STARTS WITH NEW LINE
015C                    OUTBUF  RB      62          ; BUFFER FOR MOVED CHARS
019A 00                         DB      0
                                END
```

END OF ASSEMBLY. NUMBER OF ERRORS: 0. USE FACTOR: 0%

```
; Ausgabe der von ASM86 erzeugten Symboldatei

B>TYPE SAMPLE.SYM


0000 VARIABLES
011B CCOUNT     011C INBUF      011A INPLINE    0100 MESSAGE    015C OUTBUF
015A PRINTL

0000 NUMBERS
000D CR         0024 DOLLAR     000A LF         0009 PSTRING    0000 RESET
000A RSTRING

0000 LABELS
001E ACCEPT     002C BACKST     0029 BDOS       0024 EXIT       0018 PRINT
0000 START      0037 STORE


; Bilden einer Kommandodatei

B>GENCMD SAMPLE

BYTES READ    006B
RECORDS WRITTEN 05


; Versuch, das Programm im Direktlauf abzuarbeiten

B>SAMPLE

SID86 SAMPLE SESSION

12345
5432


; Das Ergebnis ist nicht korrekt.  Das erste Zeichen fehlt.
; Laden des ausführbaren Programms und der Symboldatei zum Test.
```

```
  SID86 SAMPLE SAMPLE
SID86 V 1.0
     START      END
CS 7BB7:0000 7BB7:004F
DS 7BBC:0000 7BBC:019F
SYMBOLS

; Ausgabe aller geladenen Symbole.

#H
0000 VARIABLES
011B CCOUNT
011C INBUF
011A INPLINE
0100 MESSAGE
015C OUTBUF
015A PRINTL
0000 NUMBERS
000D CR
0024 DOLLAR
000A LF
0009 PSTRING
0000 RESET
000A RSTRING
0000 LABELS
001E ACCEPT
002C BACKST
0029 BDOS
0024 EXIT
0018 PRINT
0000 START
0037 STORE

; Reassemblieren des Beginns vom Code-Segment.

#L
START:
  7BB7:0000 MOV   DX,0100 .MESSAGE
  7BB7:0003 CALL  0018 .PRINT
  7BB7:0006 MOV   DX,011A .INPLINE
```

```
  7BB7:0009 CALL  001E .ACCEPT
  7BB7:000C CALL  002C .BACKST
  7BB7:000F MOV   DX,015A .PRINTL
  7BB7:0012 CALL  0018 .PRINT
  7BB7:0015 JMP   0024 .EXIT
PRINT:
  7BB7:0018 MOV   CL,09 .PSTRING
  7BB7:001A CALL  0029 .BDOS
  7BB7:001D RET
ACCEPT:
  7BB7:001E MOV   CL,0A .LF

; Testen der ersten beiden Befehle des Programms.

#T2
          AX   BX   CX   DX   SP   BP   SI   DI   IP
--------- 0000 0000 0000 0000 092C 0000 0000 0000 0000 MOV   DX,0100 .MESSAGE
--------- 0000 0000 0000 0100 092C 0000 0000 0000 0003 CALL  0018 .PRINT
*7BB7:0018 .PRINT

; Weiter im Direktlauf mit einem Unterbrechungspunkt nach der Bedienereingabe.

#G,.BACKST
SID86 SAMPLE SESSION

12345
*7BB7:002C .BACKST

; Anzeige des Status

#X
          AX   BX   CX   DX   SP   BP   SI   DI   CS   DS   SS   ES   IP
--------- 0000 0000 050D 0102 092A 0000 0000 0000 7BB7 7BBC 0100 0100 002C
MOV    DI,0000 .VARIABLES

; Ausgabe des Eingabepuffers.

#D.INBUF,+7
7BB7:011C 31 32 33 34 35 00 00 00 12345...
```

```
; Die eingegebenen Zeichen stehen richtig im Eingabepuffer,
; weiter einige Befehle der Umspeicher-Routine testen.

#T5
           AX   BX   CX   DX   SP   BP   SI   DI   IP
--------- 0000 0000 050D 0102 092A 0000 0000 0000 002C MOV   DI,0000 .VARIABLES
--------- 0000 0000 050D 0102 092A 0000 0000 0000 002F MOV   SI,[011B] .CCOUNT =3105
--------- 0000 0000 050D 0102 092A 0000 3105 0000 0033 AND   SI,00FF
STORE:
-------P- 0000 0000 050D 0102 092A 0000 0005 0000 0037 MOV   AL,011C[SI] .INBUF =00
-------P- 0000 0000 050D 0102 092A 0000 0005 0000 003B MOV   015C[DI].AL .OUTBUF =00
*7BB7:003F

; Hier läuft etwas verkehrt. Das letzte eingegebene Zeichen müßte
; jetzt in AL stehen, AL ist aber 0.

#D.INBUF+SI,+0
7BBC:0121 00 .

; Die Indexierung ist offenbar fehlerhaft.
; Der Index SI müßte von N - 1 bis 0 laufen.
; Es kann auch die Basisadresse um 1 verringert werden.
; Test der Indexierung.

#D.INBUF-1+SI,+0
7BBC:0120 35 5

; Jetzt ist der Zugriff korrekt. Der Fehler soll in der CMD-Datei
; korrigiert und in der Datei fixiert  werden.
; Dazu wird die gesamte Datei einschließlich Header gelesen.

#RSAMPLE.CMD
  START      END
7B8F:0000 7B8F:027F

; Der Header verschiebt die Paragraphgrenze des Code-Segments um 8.

#H7B8F,8
+ 7B97   - 7B8F   * 0003DC78   / 0F71 (0007)
```

```
; Ab 7B97 beginnt der Code.
; Wenn diese Adresse als Basisadresse für das L-Kommando
; genommen wird, sind alle Symbole wieder korrekt.

#L7B97:0
START:
  7B97:0000 MOV    DX,0100 .MESSAGE
  7B97:0003 CALL   001B .PRINT
  7B97:0006 MOV    DX,011A .INPLINE
  7B97:0009 CALL   001E .ACCEPT
  7B97:000C CALL   002C .BACKST
  7B97:000F MOV    DX.015A .PRINTL
  7B97:0012 CALL   0018 .PRINT
  7B97:0015 JMP    0024 .EXIT
PRINT:
  7B97:001B MOV    CL,09 .PSTRING
  7B97:001A CALL   0029 .BDOS
  7B97:001D RET
ACCEPT:
  7B97:001E MOV    CL,0A .LF

; Reassemblieren der Umspeicher-Routine

#L.STORE,+7
STORE:
  7B97:0037 MOV    AL,011C[SI] .INBUF
  7B97:003B MOV    015C[DI],AL .OUTBUF

; Der Befehl auf Offset 37 soll korrigiert werden.

#A37
7B97:0037  MOV AL,.INBUF-1[SI]
7B97:003B  .
#WSAMPLE.CMD

; Die korrigierte Datei wird auf die Platte zurückgeschrieben
; und zum weiteren Test erneut geladen.

#ESAMPLE.CMD
```

```
     START     END
CS 7BB7:0000 7BB7:004F
DS 7BBC:0000 7BBC:019F

#G,.BACKST
?
#

; Hier wurde vergessen, die Symboldatei neu zu laden.
; Wiederholen des Ladekommandos.

#ESAMPLE SAMPLE
     START     END
CS 7BB7:0000 7BB7:004F
DS 7BBC:0000 7BBC:019F
SYMBOLS

; Testen einiger Befehle ohne Protokollierung der Unterprogramme.

#TW4
           AX   BX   CX   DX   SP   BP   SI   DI   IP
-------P- 0000 0000 050D 0102 092C 0000 0005 0000 0000 MOV   DX,0100 .MESSAGE
-------P- 0000 0000 050D 0100 092C 0000 0005 0000 0003 CALL  001B .PRINT
SID86 SAMPLE SESSION


-------P- 0000 0000 0119 0102 092C 0000 0005 0000 0006 MOV   DX,011A .INPLINE
-------P- 0000 0000 0119 011A 092C 0000 0005 0000 0009 CALL  001E .ACCEPT12345
*7BB7:000C

; Weiter in der Umspeicher-Routine testen.

#T2
           AX   BX   CX   DX   SP   BP   SI   DI   IP
-------P- 0000 0000 050D 0102 092C 0000 0005 0000 0000 CALL  002C .BACKST
BACKST:
-------P- 0000 0000 050D 0102 092A 0000 0005 0000 002C MOV   DI,0000 .VARIABLES
*7BB7:002F
```

```
; Reassemblieren einiger Befehle.

#L
  7BB7:002F MOV    SI,[011B] .CCOUNT
  7BB7:0033 AND    SI,00FF
STORE:
  7BB7:0037 MOV    AL,011B[SI] .CCOUNT
  7BB7:003B MOV    015C[DI],AL .OUTBUF
  7BB7:003F INC    DI
  7BB7:0040 DEC    SI
  7BB7:0041 JNZ    0037 .STORE
  7BB7:0043 MOV    BYTE 015C[DI],24 .OUTBUF
  7BB7:0048 RET
  7BB7:0049 ADD    [BX+SI],AL
  7BB7:004B ADD    [BX+SI],AL
  7BB7:004D ADD    [BX+SI],AL

; Die Korrektur auf Offset 37 ist korrekt. CCOUNT entspricht dem Wert
; des Ausdruckes INBUF-1. Zum vollständigen Test der Umspeicher-
; Routine wird ein Protokollierpunkt auf den Endetest gesetzt.

#P41,FFFF

; Weiter im Direktlauf mit Protokollierpunkt bis vor die Ausgabe.

#G,.PRINT

FFFF PASS 7BB7:0041
--------- 0035 0000 050D 0102 092A 0000 0004 0001 0041 JNZ    0037 .STORE
FFFE PASS 7BB7:0041
-------P- 0034 0000 050D 0102 092A 0000 0003 0002 0041 JNZ    0037 .STORE
FFFD PASS 7BB7:0041
--------- 0033 0000 050D 0102 092A 0000 0002 0003 0041 JNZ    0037 .STORE
FFFC PASS 7BB7:0041
--------- 0032 0000 050D 0102 092A 0000 0001 0004 0041 JNZ    0037 .STORE
FFFB PASS 7BB7:0041
-----Z-P- 0031 0000 050D 0102 092A 0000 0000 0005 0041 JNZ    0037 .STORE
*7BB7:0018 .PRINT
```

```
; Die Umspeicherung arbeitet offenbar korrekt. In AL stehen nacheinander
; die eingegebenen Zeichen. Die Indizes in SI und DI laufen korrekt.
; Nochmals Anzeige der eingegebenen und umgespeicherten Zeichenkette.

#D.INBUF,+7
7BBC:011C 31 32 33 34 35 00 00 00 12345

#D.OUTBUF,+7
7BBC:015C 35 34 33 32 31 24 00 00 54321¤..

; Abarbeitung ohne Unterbrechung bis zum Ende.

#G
54321

; Die Ausgabe ist korrekt. Das Programm arbeitet fehlerfrei.
; Die Korrektur sollte jetzt im Quellprogramm ausgeführt werden
; und nochmals eine Kommandodatei durch GENCMD gebildet werden.
```

Anlage Fehlermitteilungen von SID86

AMBIGOUS OPERAND

Mehrdeutiger Operand. Zu einem A-Kommando wurde ein mehrdeutiger Operand angegeben. Der Operand muß durch BYTE oder WORD spezifiziert werden.

BAD FILE NAME

Fehlerhafter Dateiname. Ein Dateiname in einem E-, R- oder W-Kommando ist fehlerhaft.

BAD HEX DIGIT

Fehlerhafte hexadezimale Ziffer. Eine Datei vom Typ SYM hat eine fehlerhafte hexadezimale Ziffer.

CANNOT CLOSE

Datei kann nicht geschlossen werden. Eine Plattendatei, die durch ein W-Kommando geschrieben wurde, kann nicht geschlossen werden.

DISK READ ERROR

Plattenlesefehler. Eine Plattendatei, die in einem R-Kommando spezifiziert wurde, kann nicht gelesen werden.

DISK WRITE ERROR

Plattenschreibfehler. Eine Schreiboperation auf die Platte während eines W-Kommandos kann nicht richtig ausgeführt werden. Vermutlich ist die Platte voll.

INSUFFICIENT MEMORY

Unzureichender Speicher. Es ist nicht genügend Speicher vorhanden, um eine Datei durch ein R- oder E-Kommando zu laden.

MEMORY REQUEST DENIED

Speicheranforderung abgelehnt. Eine Speicheranforderung während eines R-Kommandos wurde abgelehnt. Es können nur bis zu sieben Speicherbereiche unter SID86 gleichzeitig verwaltet werden.

NO FILE

Keine Datei. Die in einem R- oder E-Kommando spezifizierte Datei kann nicht auf der Platte gefunden werden.

NO SPACE

Kein Speicherbereich. Ein W-Kommando kann nicht ausgeführt werden, weil das Dateiverzeichnis voll ist.

SYMBOL LENGTH ERROR

Symbolfehler. Ein Symbol der SYM-Datei, die durch ein E-Kommando geladen wurde, hat mehr als 31 Zeichen.

SYMBOL TABLE FULL

Symboltabelle ist voll. Es ist kein Platz mehr in der Symboltabelle von SID86.

VERIFY ERROR AT s:o

Vergleichsfehler. Der durch ein F-, S-, M- oder A-Kommando eingegebene Wert kann nicht zurückgelesen werden. Das kann von einem fehlerhaften RAM herrühren, oder es wurde versucht, auf einem ROM zu schreiben, oder der Speicher existiert nicht im angegebenen Speicherbereich.

Sachwortverzeichnis

A-Kommando 14
Adresse 7
Assembler-Anweisung 35
Ausdruck 8

B-Kommando 15
Basisseite 20
Bezug, symbolischer 10

D-Kommando 15
Dateisteuerblock 20
Durchlaufzählung 23

E-Kommando 16

F-Kommando 18
Flags 30
Funktion, arithmetisch 19

G-Kommando 18

H-Kommando 19

I-Kommando 20
Interrupt-Behandlung 7

Kommando 6
Kommandoanforderung 5f.
Kommandoparameter 20
Kommandoparameter-Puffer 20
Kommandozeile 6

L-Kommando 21
Label 21
List-Adresse 21
Literal, alphanumerisch 9
Literal, dezimal 8
Literal, hexadezimal 8

M-Kommando 22

Operator 11

P-Kommando 22
Programmschritt 27
Protokollierpunkt 22

R-Kommando 24

S-Kommando 26
Segmentwert 7, 33
Standardsegmentwert 33
Startadresse 18
Symbol, ausgewähltes 11
Symboldatei 5
Symboltabelle 10

T-Kommando 27
Typ1-Segmentwert 33

C 1015-0004-1 M 3030

Typ2-Segmentwert 33

U-Kommando 29
Unterbrechungspunkt 18

V-Kommando 29

W-Kommando 30

X-Kommando 30

Zeichenkette 9

Ag 706/209/87 – 4119